CPF Data Management System

Requirements

Workflow Processing Requirements

The intent of workflow processing is to automate a sequence of steps (tasks) that are required to run in response to an input condition (event). In the domain of CPF, for example, performing inter-calibration tasks for an event in response to the arrival of the required CPF L1 and CERES/VIIRS input files.

A good workflow processing system is general-purpose and can support many different use-cases. We require a general tool so that all current and future CPF processing can be supported with minimal code changes.

Concrete CPF inter-calibration use-cases. These are designed with the aid of subject matter experts.

Workflows

- Workflows consist of multiple tasks (i.e. science jobs) chained together.
- Workflows (and associated objects) must be represented as an XML file for simple viewing and editing
- Workflows (and associated objects) must be stored in a repository to facilitate software-based editing and viewing

Workflow Execution

- Incoming events trigger the start of workflows
- Workflow tasks may run sequentially or in parallel.
- Workflow tasks must execute on a Univa Grid Engine (UGE) cluster
- Workflow tasks must not over-consume resources in the UGE cluster.
  - Using UGE load balancing and/or load monitoring features.
- Instances of executing workflows must be defined in a repository, including the current state of all workflow tasks

Workflow Events

- Arrival of new data file(s) can source new events
  - Files can be discovered by scanning folders
  - Files can be discovered by Email notification of an archive system
- The calendar can source new events
  - Calendar events can be defined in a CRON schedule
- Human operators can source new events

Workflow Tasks

- Tasks have preconditions and postconditions.
  - Preconditions must be true before a task may run
  - Postconditions must be true after a task has run
- Task preconditions must include:
  - Input files exist and are valid
  - Metadata exists and is valid
- Task postconditions must include:
  - Output files exist and are valid
  - Metadata exists and is valid
  - Valid exit codes
- Tasks must be stored in a repository and include relevant metadata:
  - Description
  - Preconditions & postconditions
  - Metadata including options and default values

    - Logging configuration
    - Version information
- Workflow tasks may be stand-alone executables or Java objects that implement an interface.
- Workflow tasks may run inside a software container (i.e. docker) to help simplify deployment
- Workflow tasks may obtain additional run-time support via a Task Support Library

Workflow Task Support Library

- Workflow tasks written in Java or any language with "C" binding (i.e. most languages) may utilize a Task support library
    - High priority language bindings to include C++ and Fortran
    - Lower priority language bindings to include Python and Matlab
- The task support library is separate from HDF APIs
    - Tasks written in Java may use the HDF Object package or the HDF4/HDF5 JNI libraries to read/write HDF4 and HDF5 files.
    - Tasks written other languages may also use HDF5 and HDF4 libraries (as available).
- Task support library must provide centralized logging.
    - Log entries are task & time stamped and collected in a central repository for further filtering and viewing
    - Logging to include standard levels (TRACE, DEBUG, INFO, WARNING, ERROR) and hierarchical loggers.
    - Logging must be asynchronous and have a minimal impact on performance
- Task support library may include other routines to facilitate or standardize common IC tasks (TBD)
- Task support library must provide read/write access to workflow metadata

Workflow Metadata

- Workflow metadata may provide additional context to an executing workflow and its tasks such as:
    - date/time stamps
    - user options
    - other information that is either inconvenient or inappropriate to store in data files.
- Metadata may take the general form "key=value".

Webserver

- CPF must employ a web server to support validation & operations
- High priority: workflow instance monitoring
    - Show state of workflow execution
    - Show workflow and/or task errors
    - View logs and filter by workflow and task
- High priority: workflow instance control
    - Pause and resume workflow execution (*TBD, may not be realistic for CPF*)
    - Stop all workflow execution and save current state to repository (*TBD, may not be realistic for CPF*)
    - Reset all workflows (i.e. restart from the beginning) (*TBD, not realistic be for CPF*)
    - Test workflows
- High priority: monitoring events
    - Ability to see available event sources (file scanner, etc) and recent activities
- Low priority: workflow visualization and configuration
    - Graphical representation of workflow and state of workflow tasks
    - Ability to edit workflow and workflow tasks
    - Ability to edit workflow event sources
- Low priority: performance & UGE integration
    - Link to or incorporate UGE cluster web pages
    - Show task performance statistics

Inter-calibration Requirements

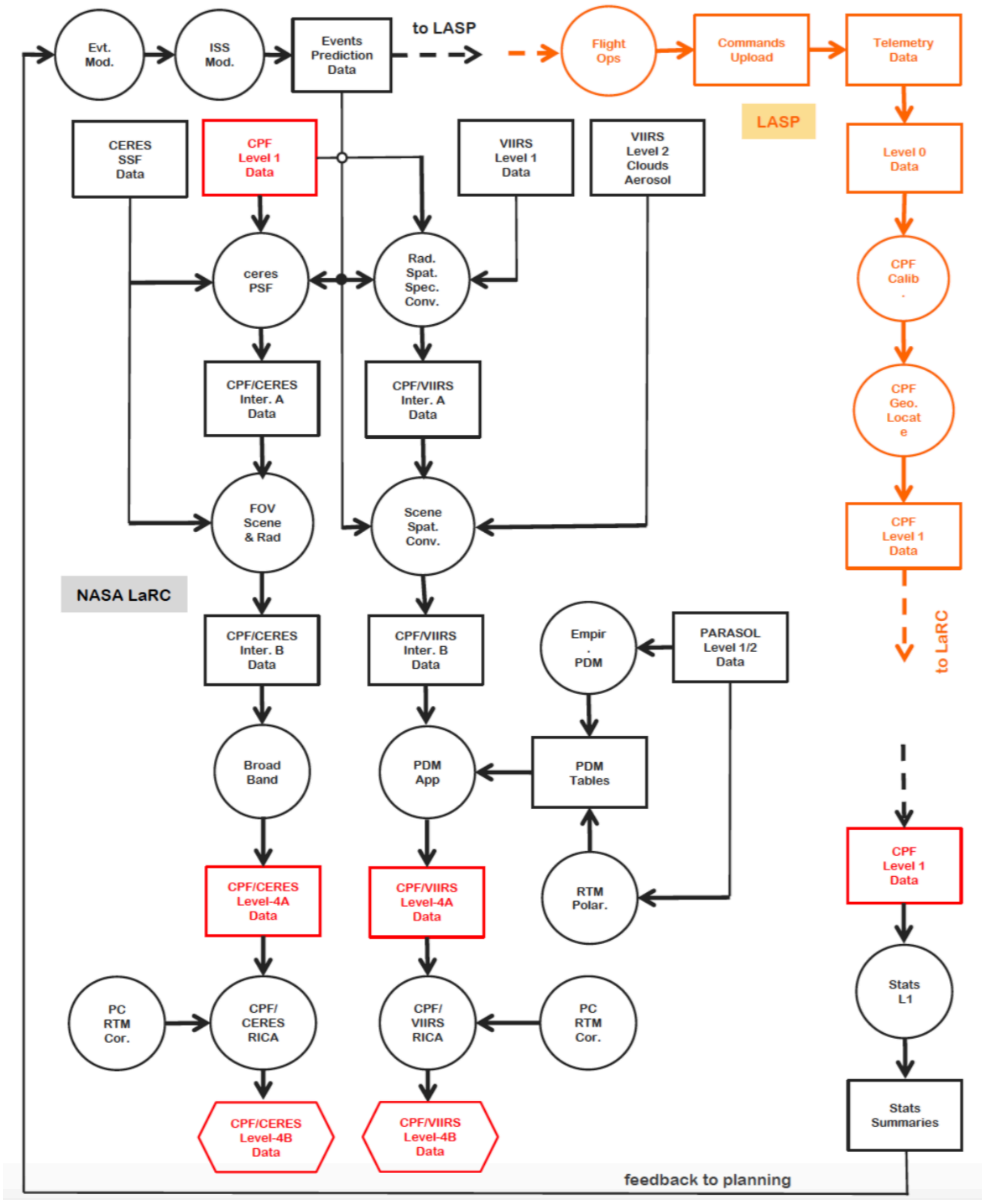
Inter-calibration Workflows

The following analysis diagram helps us understand the anticipated inter-calibration workflows:

- Circles represent workflow tasks and rectangles represent workflow data.
- Orange shapes are tasks/data belonging to LASP and not part of this system.
- Red rectangles represent archived data products (archive submission tasks are not currently shown)
- Rectangles flowing into a task are task preconditions
- Rectangles flowing out of a task are task postconditions and can be modeled as data files stored on the cluster or possibly as workflow metadata.

Events are not depicted in this diagram. Anticipated events are:

- A new CPF L1 file arrives from LASP
- A new CERES SSF data file is discovered
- A new VIIRS L1 or L2 Cloud/Aerosol data file is discovered



Workflow Tasks

The following concrete workflow tasks are required.

CPF IC Scheduler

- Generates CPF inter-calibration Schedule for CERES/VIIRS vs. ISS
- TBD event prediction requirements, which may be different that standard MIIC LEO/LEO
- TBD if this prediction is done by MIIC or other software

ISS Modeling

- Alters CPF IC Schedule to comply with ISS
- TBD ISS Modeling requirements (ISS schedule, orbit changes, instrument occlusion, etc)
- TBD software language (may include STK simulation)

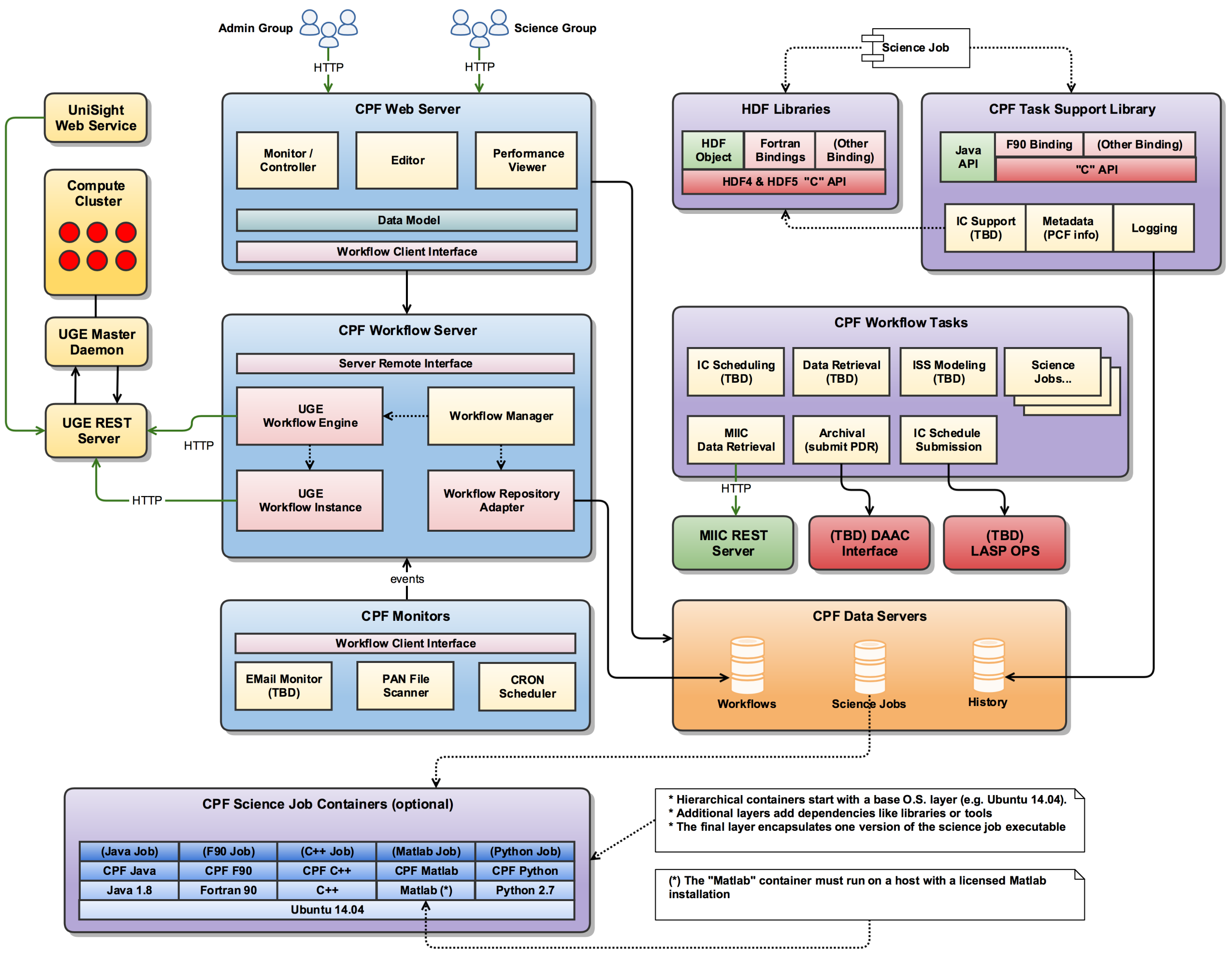
MIIC IC Data Retrieval

- Must locate and retrieve CERES and VIIRS data required for inter-calibration tasks
- (TBD) MIIC IC Events may not be predicted, but instead provided a-priori to match duration of CPF L1 files
- Must support these CERES/VIIRS data collections:
    - CERES SSF
    - VIIRS L1
    - VIIRS L2 Cloud & Aerosol
- Must accept list of required data variables
- Must stage data to cluster to be used as science job input

CPF IC Schedule Submission

- Task to send CLARREO IC Schedule to LASP

Architecture



* Hierarchical containers start with a base O.S. layer (e.g. Ubuntu 14.04).
* Additional layers add dependencies like libraries or tools
* The final layer encapsulates one version of the science job executable

(*) The "Matlab" container must run on a host with a licensed Matlab installation

CPF Workflow Server

The workflow server is a general-purpose tool to execute pre-defined workflows in response to incoming events. We will re-use an existing open-source workflow system that meets our requirements, as opposed to writing this component from scratch. CPF inter-calibration workflows and events are defined using the data model of the workflow system. This will both save development time and provide a more robust system.

The workflow server will be a separate server process that deploys to same node as the UGE Master Daemon and UGE REST Server. Running on the cluster head node provides better performance and safety since the UGE job queue will only be accessed locally. The workflow server will run in its own Java VM, isolating it from other non-critical components.

As it is a standalone server it must provide a remote interface. Clients use this interface to send events, retrieve workflow status, and to control workflows (i.e. start, stop, pause, etc).

Workflow Manager

The primary component of the workflow server is the workflow manager, which loads workflow definitions from some repository, process incoming events, and of course manages the execution of all workflows and their constituent tasks. Its main job is to kick off workflows in response to incoming events. It also executes the individual tasks comprising a workflow as directed, checks task preconditions and postconditions, handles errors, processes operator inputs, etc.

Task Execution Abstractions

Because workflow tasks must run on compute resources, the workflow manager provides abstractions for the *execution* and *management* of a task. A *Workflow Engine* represents a resource that can execute a task. A *Workflow Instance* represents a running task which may be queried as to its status, or killed. Since our implementation uses UGE we need a *UGE Workflow Engine* and a *UGE Workflow Instance* as shown in the architecture diagram.

Workflow Repository

Objects comprising a workflow will be stored in an external repository and loaded on demand. The workflow data model is of course determined by the workflow software we use. We use their workflow data model to implement the required CPF workflows. As we gain experience we may find opportunities to modify or adapt this model to better suit our needs.

The workflow manager accesses the workflow repository through an interface shown in the architecture diagram as the *Workflow Repository Adapter*. This will allows us to replace or update the workflow repository when necessary. Initially, our workflow repository will simply be a collection of XML files. Later, we may integrate database storage of workflows to facilitate web-based workflow tools.

CPF Monitors

Monitors are stand-alone daemons that check for meaningful conditions (e.g. the availability of a new CPF L1 file), and then send the appropriate event to the Workflow Server. They should be separate from the workflow server since performing this activity is orthogonal to workflow processing. Monitor daemons may also need to be deployed to a different host than that running the workflow server. For example, we may need to scan for a PAN (product availability notice) files that are deposited to an FTP server outside the LaRC firewall.

Having separate monitor daemon(s) introduces a new point of failure to the system. There the CPF Monitor will send periodic heartbeat events to the Workflow Server. This allows our system to know which monitors are alive and to help diagnose problems.

Types of monitors

We will build a file-based monitor that will simply scan a folder for files matching a regular expression. These files are commonly used to signal the presence of new data available for inter-calibration workflows.

We will also build a CRON-based monitor – this is useful for send heartbeat events or any other workflows that must run periodically (e.g. housekeeping type tasks like removing temporary files).

If necessary, we will also build an Email-Based monitor. Some archive systems send Email to notify when a file has been successfully archived or an order is ready for pick-up.

CPF Data Servers

Like most systems, we will need databases or other types of repositories to store common system data. Note that the implementation of a data server can change over time. We may start with "repositories" that are simply folders on a shared network drive. Later, these may be replaced with databases or other open-source repositories that provide more features.

Workflow Server

Science Job Server

The science job server is a repository to store executables that run as part of CPF inter-calibration workflows.

Flat-file Repository

Initially, the simplest repository is a folder on a shared network drive mounted on all cluster nodes, along with a naming convention to help organize multiple versions of jobs and their supporting files. We will use separate folders to distinguish jobs under test and those in production. Every unique version of every job occupies its own folder. All jobs are started by a "go.sh" script in this folder, along with any other supporting files:

- `/cpf_jobs/test/[job_name]/[job_version]/go.sh`
- `/cpf_jobs/production/[job_name]/[job_version]/go.sh`

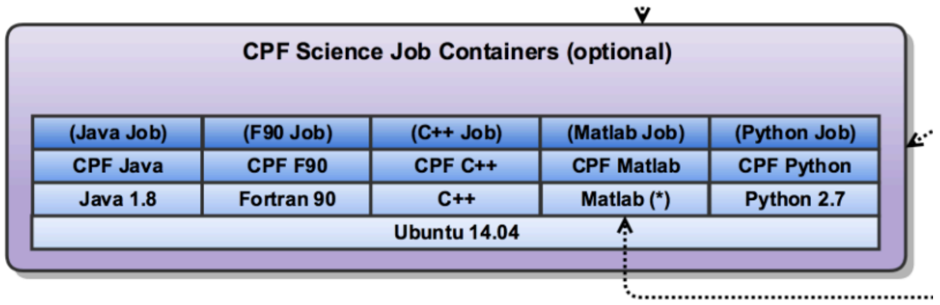
Software Container Repository

The science job server may also be implemented as a software container repository using Docker. Software containers encapsulate the science job and all of its system dependencies. The Univa Grid Engine is integrated with Docker and is able to run jobs as docker containers.

The use of docker containers has the following implications:

- Once created, a science job "container" can run on any host, without having to first configure the host computer with required interpreters, system libraries, etc. This makes it possible to run science jobs in the cloud, and to run science jobs that require a nonstandard environment.
- Containerized software is safer since it is not able to access host resources unless explicitly authorized. For example, it can't read or modify host files or accept incoming network connections.
    - Access to local cache or network drives must be explicitly configured at container start.
- Containers are ephemeral so any changes made inside the container itself (modified files, configuration changes, etc.) are gone when the container exits
    - We will provide all containers with a folder where files that must persist after exit can be placed. This is where log files should be written.

Docker also provides a docker "image" repository. An image is a disk file that can be used to start new running containers. Images are additive and are created by starting with one version of a base image and then installing additional software or libraries. One possible design of CPF docker images is shown here:



All images start with a large Ubuntu operating system image. Images build on the base OS by providing system support for the major types of software environments we need to support: Java 1.8, Fortran 90, C++, Python 2.7, Matlab, etc. Note that these images are created and maintained by third parties so it is unlikely we will want or need to create our own.

> ⭐ Since Matlab is a licensed product the "Matlab" image links to Matlab installed on the host computer. Therefore any Matlab-based jobs must run on a host where Matlab is already installed and licensed.

The next layer of images (CPF Java, CPF F90, etc.) adds common libraries that all CPF science jobs need. This is where the CPF Task Support Library is installed. Since Docker images are versioned this allows the CPF image layer to evolve independently from the science job image layer. For example, suppose the CPF Task Support Library for Java undergoes a major change from version 1.1 to version 2.0. Science jobs using old version can list their base image as "CPF_Java:v1.1". Science jobs that are compatible with the new version can list their base image as "CPF_Java:v2.0" or "CPF_Java:latest".

The final layer of images add the science jobs themselves. Of course these are also versioned, so workflows can reference "VIIRS_spatial_convolution:v3.3" or "VIIRS_spatial_convolution:latest".

History Server

The history server is a common repository to store all actions taken by the CPF system. Ultimately, this includes a record incoming events and the workflows that were executed in response. This provides traceability to know exactly what the system did, when, and with what data.

This server also stores detailed information like the logging output of individual science jobs. Centralized logging will make it easier to monitor running workflows and diagnose problems.

CPF Task Support Library